Manuscript received Nov. 3, 2007; revised Dec. 20, 2007

FPGA Design, Implementation and Analysis of Scalable Low Power Radix 4 Montgomery Multiplication Algorithm

A. A. IBRAHIM [1], H. A. ELSIMARY [1], A. M. NASSAR [2]

[1] Electronics Research Institute, Cairo, Egypt,

[2] Cairo University, Cairo, Egypt

Abstract: - This paper proposes an efficient algorithm and Processing Element (PE) architecture for a Multiple Word Radix 4 Montgomery Modular (MWR4MM) multiplier. This architecture is developed considering an important design factor - power consumption - in addition to other design factors that is considered previously in many publications such as performance and scalability. To increase performance, we used a recoding scheme that eliminates the reduction step in the Montgomery algorithm and the PE architecture is based on the Carry-Save Adder (CSA). To achieve scalability, we implement the algorithm based on the multiple-word operation. Lastly to lower power consumption, we devised several effective techniques for reducing the glitches and the Expected Switching Activity (ESA) of high fan-out signals.

Key-words: - Montgomery multiplication, Scalability, Cryptography, Secure communications, Low power modular multipliers

1 Introduction

Nowadays, most cryptography applications use modular multiplication extensively. Many cryptography applications, such as the encryption /decryption operations of the RSA algorithm [1], the Digital Signature Standard [2], the Diffie-Hellman key exchange algorithm [3], and elliptic curve cryptography [4], all has an extensive use of modular multiplication and modular exponentiation. The modular exponentiation operation applies modular multiplication operation repeatedly [5, 6, 7, 8, 9]. So, the performance of any cryptography application widely depends on the efficiency of the modular multiplication operation.

There are several approaches for computing the modular multiplication operation. The most efficient approach is the Montgomery modular Multiplication (MM) algorithm [10, 11]. The main advantage of this algorithm over ordinary modular multiplication algorithms is that the modulus reduction of the partial product is done by shift operations which are easy to implement in hardware.

There are several papers published about the scalable Montgomery multipliers. The most important of them that are published by A. F. Tenca and Ç. K. Koç [12, 13, 14, 15]. In their publications, they introduced what is called word-based Montgomery multiplication algorithm to implement the scalability. However, they did not consider their hardware from the low power consumption point of view.

In high radix implementation of MM algorithm, a modified Booth recoding scheme [16] is used to improve the performance by reducing the number of iterations to a number depends on the radix of computation. As the radix of computation increases as the number of iterations reduces. By using this recoding scheme, the publishers can achieve on-the-fly and simple calculation of the partial product [17].

The goal of this work is to describe a new radix-4 scalable architecture that is developed considering all kinds of design factors such as performance, scalability and power consumption.

This paper is organized as follows. Section 2 presents the proposed Multiple-Word Radix-4 Montgomery Multiplication (MWR4MM) algorithm with recoding . In this section, we also introduce a specialized recoding scheme as well as the well-known Booth recoding scheme. Section 3 presents the architecture of the modular multiplier that implements (MWR4MM) algorithm. Section 4 presents the architecture of the Processing Element (PE) of the MWR4MM algorithm. Section 5 describes several techniques to decrease power dissipation. Experimental results, with power, area,

and maximum speed are given in Section 6. Finally Section 7 concludes the paper.

2 MWR4MM Algorithm

The notation used in this paper is as follows.

- M : modulus.
- m_j : a single bit of M at position j.
- A : multiplier operand.
- B : multiplicand operand.
- n : operand's precision.
- R : a constant (called a Montgomery parameter), $R = 2^n$.
- S : intermediate partial product , or final result of modular multiplication.
- qa_j : coefficient determines the multiples of the multiplicand B ($qa_j * B$).
- qm_j : coefficient determines the multiples of the modulus M ($qm_j * M$).
- w : word size (in number of bits)of either B , M or S.
- $e = \left\lceil \frac{n}{w} \right\rceil$: number of words in either B, M or S.
- C_a , C_b : carry bits.
- $(B^{(e-1)},....,B^{(1)},B^{(0)})$: word vector of B.
- $(M^{(e-1)},....,M^{(1)},M^{(0)})$: word vector of M.
- $(S^{(e-1)},....,S^{(1)},S^{(0)})$: word vector of S.
- $S^{(i)}_{k-1....0}$: bits k - 1 to 0 from the i^{th} word of S.

Fig. 1 shows the MWR4MM algorithm. This algorithm is an extension of the Multiple-Word High-Radix ($R2^k$) Montgomery Multiplication (MW $R2^k$ MM) algorithm presented in [14], but we used a recoding scheme to recode qm_j.

we will use PP and PM to represent a partial product $(qa_j * B)$ and a multiple of modulus $(qm_j * M)$, respectively.

In the case of radix-4, qa_j, qm_j are 2-bit numbers. Thus the value sets of PP and PM are as follows:

PP $\in \{0,A,2A,3A\}$, PM $\in \{0,M,2M,3M\}$

to calculate 3A and 3M on the fly, we need two extra adders. To remove the burden of calculating 3A in the PP's value set, a modified Booth recoding scheme is popularly used. Let the $a_{j,1}$ and $a_{j,0}$ are the two bits in the j^{th} significant digit of A. Radix-4 modified Booth recoding scheme takes a bit stream $(a_{j,1}, a_{j,0}, a_{j,-1})_2$ as input and generates a recoded PP according to Table 1, where $a_{-1,1}$ is defined to be 0 and qa_j the recoded quotient digit for a PP at the j^{th} iteration.

1: $S = 0$

$a_{-1} = 0$

2: For j = 0 to n-1 step 2

3: $qa_j = Booth(a_{j+1..j-1})$

4: $(C_a, S^{(0)}) = S^{(0)} + (qa_j * B)^{(0)}$

5: $qm_j = Montg\,(S^{(0)}_{1..0} * (4 - M^{(0)^{-1}}_{1..0}) \bmod 4)$

6: $(C_b, S^{(0)}) = S^{(0)} + (qm_j * M)^{(0)}$

7: For i:= 1 to e-1

8: $(C_a, S^{(i)}) = C_a + S^{(i)} + (qa_j * B)^{(i)}$

9: $(C_b, S^{(i)}) = C_b + S^{(i)} + (qm_j * M)^{(i)}$

10: $S^{(i-1)} = (S^{(i)}_{1..0}, S^{(i-1)}_{w-1..2})$

End For;

11: $C_a = C_a \; or \; C_b$

12: $S^{(e-1)} = signext(C_a, S^{(e-1)}_{w-1..2})$

End for;

Fig. 4. Scalable MWR4MM algorithm.

Table 1. Booth recoding scheme.

Three input bits			Recoded quotient for PP	Recoded PP
$a_{j,1}$	$a_{j,0}$	$a_{j,-1}$	qa_j	PP
0	0	0	0	0
0	0	1	+1	+A
0	1	0	+1	+A
0	1	1	+2	+2A
1	0	0	-2	-2A
1	0	1	-1	-A
1	1	0	-1	-A
1	1	1	0	0

Table 2. Montgomery recoding scheme.

Three input bits			Recoded quotient for PM	Recoded PM
$sp_{0,1}$	$sp_{0,0}$	$m_{0,1}$	qm_j	PM
0	0	0	0	0
0	0	1	0	0
0	1	0	-1	-*M*
0	1	1	+1	+*M*
1	0	0	+2	+2*M*
1	0	1	+2	+2*M*
1	1	0	+1	+*M*
1	1	1	-1	-*M*

Booth recoding scheme transforms the value set of PP into {–2*A*, –*A*, 0, +*A*, +2*A*}. All elements in the set are calculated by simple operations such as bit-inversion and/or bit-shift. However, 3*M* still remains in the value set of PM, and this problem cannot be solved by the Booth recoding scheme. In this paper, we adopt a specialized recoding method [17] to transform the original value set of PM into the one that also has easily obtainable elements only. We will give this method the name "Montgomery recoding scheme". Let $(sp_{0,1}, sp_{0,0})_2$ be the 2 bits in the Least Significant Digit (LSD) of SP = S + PP and $(m_{0,1}, m_{0,0})_2$ be the 2 bits in the LSD of *M*. According to the input condition that *M* has to be odd, $m_{0,0}$ is always '1'. Then, Montgomery recoding scheme takes a bit stream $(sp_{0,1}, sp_{0,0}, m_{0,1})_2$ as input and generates a recoded PM according to Table 2, where qm_j is the recoded quotient digit for a PM at the j^{th} iteration.

Montgomery recoding scheme transforms the value set of PM into {–*M*, 0, +*M*, +2*M*}. The proof that the algorithm is still correct after applying Montgomery recoding scheme comes from the fact that $3 \equiv -1 \bmod 4$.

Due to adopting the two recoding schemes, it is easy to calculate all the elements in the value sets of PP and PM. But the recoding schemes cause one negative effect: that is, all operands except for *M* are changed to the signed numbers that are more complicated to deal with than the unsigned ones.

3 MWR4MM Overall Architecture

Fig. 2 shows the overall architecture of the Montgomery modular multiplier implementing the MWR4MM algorithm (Fig. 1). It consists of two main functional blocks: Kernel and IO blocks. Each block is separated into two parts: control and datapath. The Kernel's datapath implements the computations of the MWR4MM algorithm. The Kernel's control block supplies the control signals that synchronize the system.

The interface for the user is provided by the IO block. The implementation of this block depends on the application where the multiplier will be applied and/or the system's architecture in which the multiplier will be embedded. So, there are different ways that we can use to implement this block.

The inputs to the Kernel's datapath are *w*-bit words of *B*, *M* and *S* (represented in a Carry-Save form as *SS* and *SC*) and *k* bits of *A*. with $k = 1$ for a radix-2 , $k = 2$ for radix-4, $k = 3$ for radix-8, etc. The outputs are *w*-bit words of the new partial product *S* (*SS kout* , *SC kout*).

We use the superscript star ($*$) to indicate that the signal is one word of the corresponding vector. The signals $B^{(*)}$, $M^{(*)}$, $SS^{(*)}$ and $SC^{(*)}$ represent one word of each vector *B* , *M*, *SS* and *SC*, respectively. $B^{(*)}$, $M^{(*)}$, $SS^{(*)}$ and $SC^{(*)}$ words change every clock cycle.

3.1 Kernel Datapath

All computations of the MM algorithm are implemented by The kernel's datapath. The kernel datapath [18] is organized as a pipeline of PEs separated by registers (see Fig. 3). Each clock cycle, the kernel datapath gets as inputs one word of *B*, *M*, *SS* and *SC*. Additionally, it gets (NS**k*) bits of *A* over (2*NS) clock cycles, where NS represents the number of stages in the pipeline. A stage consists of a PE and a register as shown in Fig. 3. Every second clock cycle, *k* bits of *A* are loaded in a stage. Every clock cycle, the kernel datapath outputs one word of each *SS* and *SC*. The outputs are named $SS^{(*)}kout$ and $SC^{(*)}kout$.

4 MWR4MM PE Architecture

Fig. 4 shows the block diagram of radix-4 PE. The PE is divided into two sections. The first section computes only the first 2 Least Significant Bits (LSBs) of each word of $S + qa_j * B$. One can observe that qm_j depends on 2 LSBs of the partial product *S* from the previous computational cycle, $S^{(0)}_{1..0}$, the 2 LSBs of $B^{(0)}$, and the recoded

multiplier digit qa_j. The word size for *S* needs to be at least 4 bits [14], since the 2 LSBs of *S* for the next pipeline stage will be available well before the whole word $S^{(0)}$ is available, and so they can be used in determining qm_j for the next computation.

The second section completes the computation of the word bits of $S + qa_j * B$ and the addition of full words of $S + qm_j * M$.

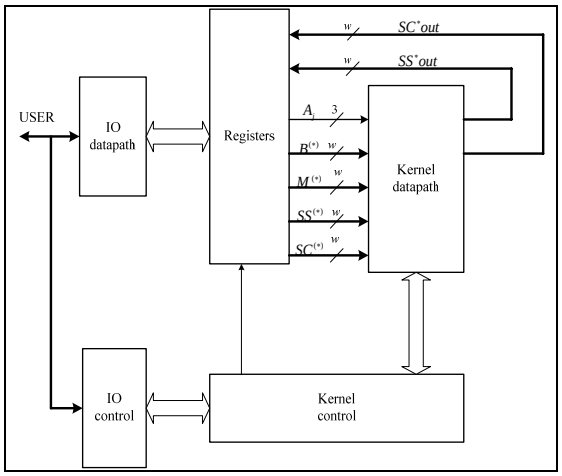

Fig. 2. Overall architecture of MWR4MM.

The computation done on the LSBs by the first section is also done for all the other remaining operand words. So, while the leftmost adder works on the LSBs of a word of $S + qa_j * B$, the topmost adder (after the input register) works on the other bits of the same word, therefore, there is one clock cycle difference between the two circuits.

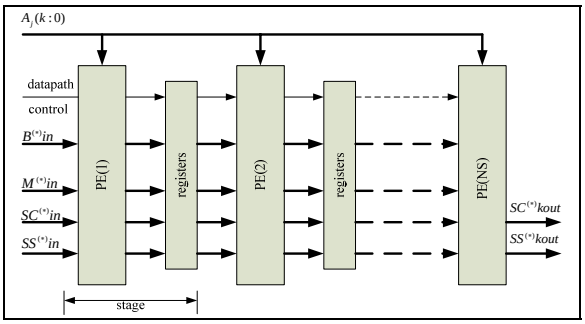

Fig. 3. Kernel datapath.

5 Low Power Techniques

In this section we further improve our hardware to dissipate less power than the one implemented directly. Inevitably all digital devices have spurious transitions or glitches internally due to unbalanced path delays, which causes worthless dynamic power dissipation. Furthermore, if fan-outs of the glitchy signals are big, then the amount of worthlessly dissipated power is significant. Such signals are the outputs from the two circuit modules in charge of the Booth and Montgomery recordings. Because the modules comprise only combinational logic circuits according to Table 1 and Table 2, their outputs must have glitches. Also, the fan-outs of the outputs are *w* that is usually very large number. To reduce the glitching power dissipation, we put in some latches and force the outputs to pass through latches. If all flip-flops and registers capture their inputs at the clock's rising edge, then the latches are transparent when the clock is in a low state. If the outputs of the two recoding modules can reach their stable values before the clock's falling edge, none of the glitches can propagate to the fan-out modules driven by the outputs. We name these latches "glitch blockers". The glitch blockers are also very effective for reducing the glitches appearing in the CSA since they synchronize the arrival of PP and PM at the CSA's inputs.

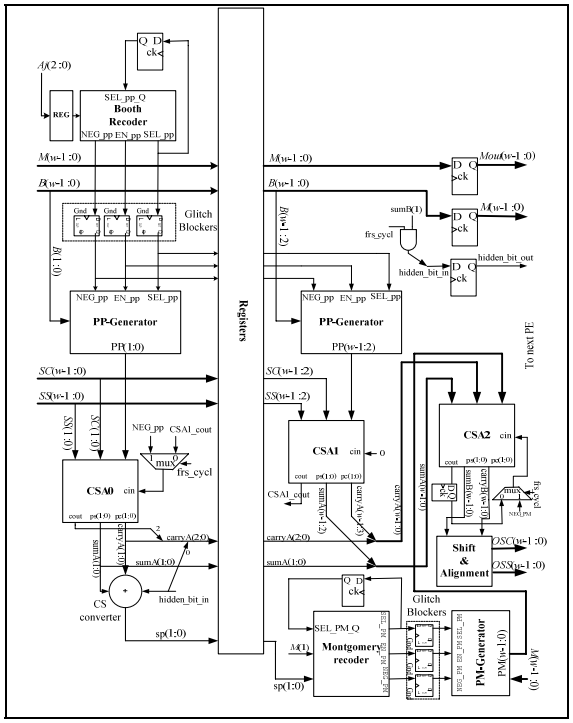

Fig. 4. Radix-4 PE architecture.

PP generator makes a PP by modifying a word of *B* according to the Booth recoder's outputs, SEL_PP and EN_PP. PM generator makes a PM by modifying a word of *M* according to the Montgomery recoder's outputs, SEL_PM and EN_PM. Several glitch blockers are located at the outputs of the two recoders.

When PP is zeroed by EN_PP, PP outputted from the PP generator does not depend on SEL_PP. Thus,

keeping SEL_PP frozen at that time is effective for reducing power dissipation. Same reasoning also applies to SEL_PM. We place two 1-bit flip-flops and construct feedback loops for SEL_PP and SEL_PM to implement this idea.

Booth recoding scheme has a feature that $+2B$ cannot follow $+B$ or $+2B$ and $-2B$ cannot follow $-B$ or $-2B$. Utilizing this feature to decrease power consumption, we suggest a binary coding method for SEL_PP as follows:

SEL_PP for $+B$ = ~(SEL_PP for $+2B$)

SEL_PP for $-B$ = ~(SEL_PP for $-2B$)

where '~' denotes bit-inversion. This binary coding method minimizes the ESA(Effective Switching Activity) of SEL_PP.

6 Experimental Results and Analysis

6.1 Synthesis and simulation environment

The experimental data, for area and time, presented in this paper was generated by "FPFA Advantage 5.0" package from Mentor Graphics corporation. The power estimation was generated using "XPower analyzer" tool from Xilinx corporation. The target technology was set to xc3s1600e-5fg484 (Spartan 3E) FPGA. The designs compared in this paper were described in VHDL and then simulated in ModelSim for functional correctness. They were synthesized using Leonardo synthesis tool for the mentioned technology.

6.2 Comparison with previous work

In this section we will compare our proposed MWR4MM architecture - in terms of time, area, and power consumption - with previous two architectures [12, 14].

In [12] a scalable radix 2 modular multiplier is presented. Figs 5 and 6 show the total computational time - as a function of the number of stages in the pipeline (NS), as well as the word size (w) of the operands - graphs for radix 2 design and 256-bit and 1024-bit operand's precision respectively.

Figs 7 and 8 show the total computational time graphs - as a function of NS and w - for our radix 4 design and 256-bit and 1024-bit operands respectively.

In [14] a scalable radix 8 modular multiplier is presented. Figs 9 and 10 show the total computational time graphs - as a function of NS and w - for radix 8 design and 256-bit and 1024-bit operands respectively.

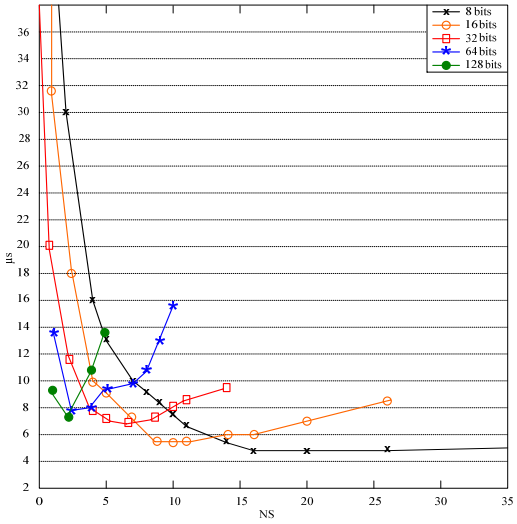

Fig. 5. Total time for radix 2 design, 256-bit operands.

Figs 5, 6,7, 8, 9, 10 show that the faster design is achieved with word size of 8 bits. NS = 16 - for all designs – is the design point at which the design is close to the fastest for 256 bit precision and is as fast as for 1024 bit operands.

The area of the PE of the three designs depends on the word size (w), and the number of stages in the pipeline (NS). The area increases as NS, and/or w of the operands increase. At w = 8, the radix 2 design PE area is 6 CLB Slices (CLBS), and our radix-4 design PE is 8 CLBS. The radix-8 design PE has area of about 20 CLBS at the same word size. The area of radix 2 and 4 are close to each other. We notice that, for a given configuration (w, NS), the area increases with the radix, i.e., A2 < A4 < A8.

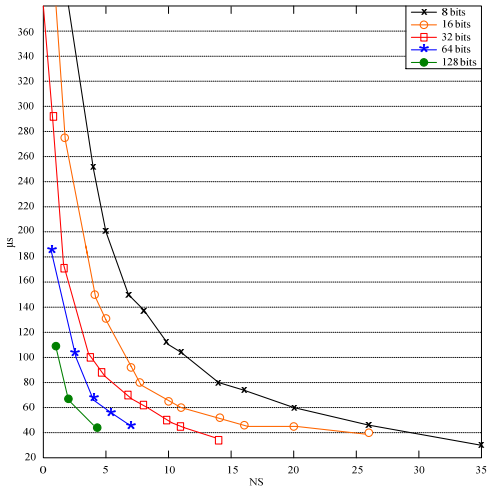

Fig. 6. Total time for radix 2 design, 1024-bit operands.

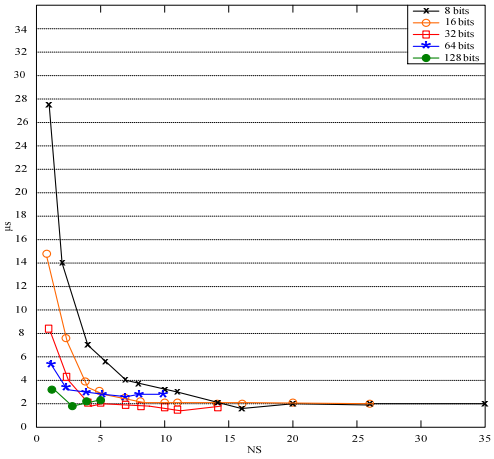

Fig. 7. Total time for our radix 4 design, 256-bit operands.

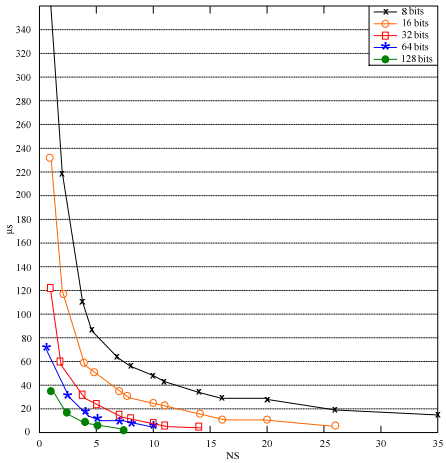

Fig. 8. Total time for radix 4 kernel, 1024-bit operands.

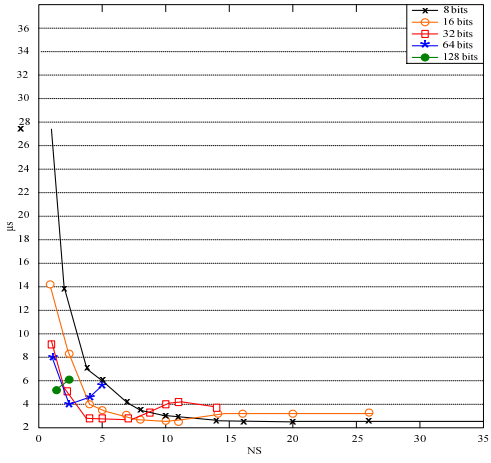

Fig. 9. Total time for radix 8 design, 256-bit operands.

Table 3 shows the total computational time in µsec for the three radices at the same area (95 CLBS). The improvement of the radix 4 design over the radices 2 and 8 designs is also shown in the Table. Other points on the figures have more gain and others have less. We conclude that our proposed radix 4 design has a significant gain in reducing the total computational time over the radices 2 and 8 designs. So, we can say that the proposed radix 4 design has the best performance among the three radices.

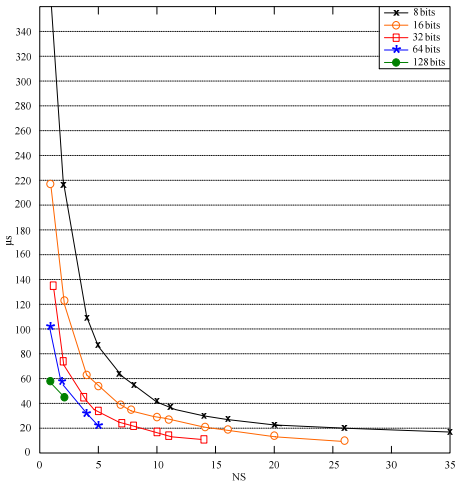

Fig. 10. Total time for radix 8 design, 1024-bit operands.

Table 4 shows the power consumption in mW for the three radices at the same area (95 CLBS). The improvement of the radix 4 design over the radices 2 and 8 designs is also shown in the table. We conclude that our proposed radix 4 design has a significant gain in reducing the power consumption over the radices 2 and 8 designs.

7 Conclusion

In this paper, we proposed an efficient architecture and implementation methodology for a scalable radix 4 Montgomery multiplier. Scalability was achieved through the multiple-word operation. High performance was achieved through a radix-4 multiplier architecture based on CSA accumulation. Applying Booth and Montgomery recoding schemes reduces the amount of hardware resources and the length of critical path. Power optimization was achieved through decreasing the glitches and the ESA of high fan-out signals. We compared the proposed scalable radix 4 architecture with other recent scalable architectures. Hardware implementation results show that our radix 4 architecture introduced in this work has a significant gain in reducing the total computation time and power consumption over the other architectures. Due to its low-power and high-performance

features, the presented Montgomery multiplier can be applicable to the mobile devices such as smart card IC and flash card IC successfully.

Table 3 Comparison between the total computation time (μsec) for the three designs taken at area (95 CLBS) with 256-bit operand precision.

	$r = 2$	$r = 4$	$r = 8$	% speed up over $r = 2$	% speed up over $r = 8$
Total Time	4.68	2.56	4.01	45%	36%

Table 4 Comparison between the power consumption (mW) for the three designs taken at area (95 CLBS) with 256-bit operand precision (@10MHZ and 1.8V).

	$r = 2$	$r = 4$	$r = 8$	% save over $r = 2$	% save over $r = 8$
Power consump.	55.9	33.23	45.12	41%	26%

References

[1] L. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Comm. ACM, Vol. 21, 1978, PP. 20-126.

[2] National Institute for Standards and Technology, Digital Signature Standard (DSS), Tech. Rep., 2000, FIPS PUB 186-200.

[3] M. Hellman, New Directions on Cryptography, IEEE Trans. on Information Theory, Vol. 22, No. 6, 1976, PP. 644-654.

[4] N. Koblitz, Elliptic Curve Cryptosystems, Mathematics of Computation, Vol. 48, No.177, 1987, PP. 203-209.

[5] A. Menezes, Applications on Finite Fields, Kluwer Academic Publishers, Boston, MA, 1993.

[6] B. Kaliski, Ç. Koç, and T. Acar, "Analysing and Comparing Montgomery Multiplication Algorithms, IEEE Micro, Vol.16, June 1996, PP. 26-33.

[7] T. Hamano, O(n)-Depth Circuit Algorithm for Modular Exponentiation , in IEEE 12th Symp. on Computer Arithmetic, IEEE Computer Society Press, Los Alamitos, CA., 1995, PP. 188-192.

[8] C. Paar, and T. Blum, Montgomery Modular Exponentiation on Reconfigurable Hardware, in IEEE 14th Symp. on Computer Arithmetic, IEEE Computer Society Press, Los Alamitos, CA., 1999, PP. 70-77.

[9] J. Bajard, L. Didier, and P. Kornerup, An RNS Montgomery Modular Multiplication Algorithm, IEEE Trans. On Computers,Vol. 47, No. 7, July, 1998, PP. 766-776.

[10] P. Montgomery, Modular Multiplication without Trial Division, Mathematics of Computation, Vol. 44, No.170, 1985, PP. 519-521.

[11] T. Yanik, E. Savas, and Ç. Koç, Incomplete Reduction in Modular Arithmetic, IEE Proc. on Computers Digital Technique, Vol. 149, No. 2, March 2002, PP. 46-52.

[12] A. Tenca, and Ç. Koç, A scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm, IEEE trans. on computers, Vol. 52, No. 9, Sept. 2003, PP. 1215-1221.

[13] A. Tenca, E. Savas, and C. Koç, A Design Framework for Scalable and Unified Architectures that Perform Multiplication in GF(p) and GF(2^m), International Journal of Computer Research, Vol. 13, No. 1, 2004, PP. 68-83.

[14] G. Todorov, A. Tenca, and Ç. Koç, High-Radix Design of a Scalable Modular Multiplier, In Ç. K. Koç, D. Naccache, and C. Paar, editors, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science No. 2162, Springer Verlag, Berlin, Germany, 2001, PP.189–205.

[15] E. Savas, A. Tenca, M. Ciftcibasi, and C. Koç, Multiplier Architectures for GF(p) and GF(2^n), IEE Proc. on Computers and Digital Techniques, Vol.151, No.2, March 2004, PP. 147-160.

[16] J. Leu, and A. Wu, Design Methodology for Booth-Encoded Montgomery Module Design for RSA Cryptosystem, Proc. IEEE Int. Symp. Circuits and Systems (ISCAS-2000), May 2000, PP. 357-360.

[17] J. Hong, and C. Wu, Radix-4 Modular Multiplication and Exponentiation Algorithm for the RSA Public-Key Cryptosystem, ASP-DAC, 2000, PP. 565-570.

[18] G. Todorov, and A, Tenca, ASIC Design, Implementation and Analysis of a Scalable High-Radix Montgomery Multiplier, Master thesis, Oregon State University, USA, 2000.